Generic Distributed Assembly and Repair Algorithms for Self-Reconfiguring Robots

Keith Kotay and Daniela Rus
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
Cambridge, Massachusetts, USA
Dartmouth Robotics Laboratory
Dartmouth College
Hanover, New Hampshire, USA
{kotay|rus}@csail.mit.edu

***Abstract*— In this paper we present generic distributed algorithms for assembling and repairing shapes using modular self-reconfiguring robots. The algorithms work in the sliding cube model. Each module independently evaluates a set of local rules using different evaluation models. Two methods are used to determine the correctness of the algorithms—a graph analysis technique which can prove the rule set is correct for specific instances of the algorithm, and a statistical technique which can produce arbitrary bounds on the likelihood that the rule set functions correctly. An extension of the assembly algorithm can be used to produce arbitrary non-cantilevered convex shapes without holes. The algorithms have been implemented and evaluated in simulation.**

I. Introduction

Current research in self-reconfiguring robots is focused on designing and building hardware, and developing algorithms coupled to specific hardware. We are interested in developing architecture-independent control and planning algorithms for such systems. In our previous work [2]–[4] we describe distributed controllers for two tasks for self-reconfiguring robots: compliant locomotion gaits and splitting a large robot with a given behavior into smaller robots with the same behavior. We demonstrate a methodology for doing this work using the sliding cube model, in which modules are represented as cubes. Each module can translate on a substrate of identical cubes and make convex and concave transitions on the substrate. The resulting algorithms are provably correct and can be instantiated easily to a wide range of physical platforms such as the Molecule and Crystal robots built in our lab [4] as well as other robot systems [8], [14], [15].

Deriving algorithms in this fashion has several advantages: (1) the algorithms are simpler in this abstract model; (2) the algorithms are easier to analyze in the abstract model; (3) the same basic algorithm can be instantiated for many different hardware types, thus providing a rigorous framework in which to compare different algorithms and hardware systems; (4) the analyses and correctness proofs will be inherited by the instantiated algorithms; and (5) ultimately this framework will lead to a better understanding of the computational problems that arise in self-reconfiguring robot research.

In this paper we extend our previous work by demonstrating distributed control algorithms for synthesizing shapes and repairing holes in them. Our approach is based on four ideas:

- Use the simplest abstraction for the robot module that fits with existing robot systems (both in shape and actuation)
- Develop distributed algorithms in the form of rules that only require local information
- Prove correctness of these algorithms with respect to the task
- Instantiate these algorithms onto real systems in a way that preserves the algorithmic properties

The use of local rules for compliant locomotion is straightforward, since locomotion does not require precise global shape control. However, it was unclear whether the exclusive use of local rules would be appropriate for assembly tasks in which a specific goal shape is required. Although each module is provided with the goal description, the possible moves are restricted to those permitted by the rule set—the goal description is only used to determine the proximity to the goal shape. As our assembly results demonstrate, it is possible to construct shapes using only local rules for a certain class of configurations. Our hole repair rule set also uses local rules to fill voids in a multilayer configuration of modules. This is accomplished by modules moving into the void and recruiting neighbor modules to follow them. Local state in the modules simulates message passing to neighbor modules which causes them to move toward the hole. For both the assembly and repair algorithms simulation is used to verify algorithmic correctness, either by graph analysis or by generating a statistical bound on the possible number of erroneous sequences of rule applications.

II. Related work

Self-reconfiguring robots were first proposed in [5]. In this planar system modules were heterogeneous and semi-autonomous. Other research focused on homogeneous systems with non-autonomous modules in two dimensions [8], [13], [15], [16] and three dimensions [11], [14], [17], [18], [21], [22]. In this type of system the modules are not capable of acting independently, and thus must remain connected. [20] is an example of a bipartite system with non-autonomous modules. Control algorithms exist for all the above implementations, although they are usually hardware specific.

Distributed control algorithms are best suited to self-reconfiguring systems, since they are more likely to scale as

the module count increases. Algorithms which require only local information are optimal, since they will require fewer communication resources. The cellular automata paradigm is well-suited for self-reconfiguring robot control, since it is an inherently distributed algorithm which uses only local information. Cellular automata has been an ongoing field of research in computer science since the early work of Stanislas Ulam who, in the 1940's, investigated the evolution of graphic constructions generated by simple rules [7]. The cellular automata paradigm has been the basis for several control methodologies [1]–[4], [6], [8], [10]. The concept and theory for a Cellular Robotic System (CRS) was proposed in [1], [6]. CRS is based on the concept of cellular automata, modified in such a way as to apply to robotic systems [1]. The individual units (modules) are simple, autonomous units. They are restricted to operating within a cellular lattice although they are not necessarily connected together to form a fixed structure. [6] describes applications and engineering problems related to cellular robotic systems.

Self-organizing collective robots which support planar self-reconfiguration are described in [8]. The modules are cubical, with four sides of the cube being used for connection between modules, and the pair of opposing sides normal to the plane of motion used for actuation. The control strategy for self-organizing collective robots is based on the cellular automata paradigm, where the local neighborhood determines module motion based on a set of rules in each module. Two reconfiguration algorithms are presented, the formation of a stair-like structure from a linear structure and the reverse. The use of module internal state information to prevent deadlock is described in the latter algorithm. Simulation of the system is done using the standard two-stage synchronous methodology for cellular automata: an evaluation stage followed by an activation stage.

Our previous work in developing generic, distributed control algorithms for self-reconfiguring robots is also inspired by cellular automata [2]–[4], [10]. Our approach is based on an abstract module instead of actual hardware in order to simplify algorithm design and analysis. We also develop proofs for the correctness of our algorithms, as well as create instantiations of the algorithms to actual hardware platforms. Instantiations allow the benefits of our provably correct algorithms to be applied to other self-reconfiguring systems, without making them system dependent. [2] presents algorithms for locomotion both with and without obstacles. [4] extends these algorithms to support climbing, turning, tunnelling, and splitting of module groups. Assembly and repair algorithms are presented in [10].

Although not based on the cellular automata paradigm, some other self-reconfiguring robot algorithms use local rules [12], [17], [19]. The 2-D Fractum reconfiguration method of [19] uses rules which specify local connection arrangements. Modules gradually accrete to the developing structure when they satisfy the connection type specified in the goal description. [12] utilizes local rules together with special messages called "scents" that decay as they propagate through the structure. Using this method, a simulation of a navigation task through an environment with obstacles is presented, including a system reconfiguration in order to pass through a narrow opening. A detailed discussion of the local rules required to accomplish the task is presented. One concern is the presence of local minima, which can prevent the task from being completed. Local rules are used to generate various locomotion gates in [17].

III. Approach

Our generic distributed approach to developing algorithms for self-reconfiguring robots has previously been described in [2], [4], [10]. The goal is to develop architecture-independent self-reconfiguring algorithms that can be instantiated to many different self-reconfiguring systems. Our approach is based on four principles:

- *a)* Work with the simplest possible abstract module, both in shape and actuation modalities
- *b)* Develop functional algorithms based on the abstract module
- *c)* Prove the correctness of the algorithms
- *d)* Instantiate the algorithms onto real self-reconfiguring systems

We have chosen to use the conceptual model of cellular automata (CA), although our system deviates from the classical CA approach in several ways. The tangible contribution of cellular automata research to our work is the use of local rules to produce global behavior. Other features of traditional cellular automata, such as non-conservation of matter and the simultaneous-update evaluation model are not appropriate for self-reconfiguring systems.

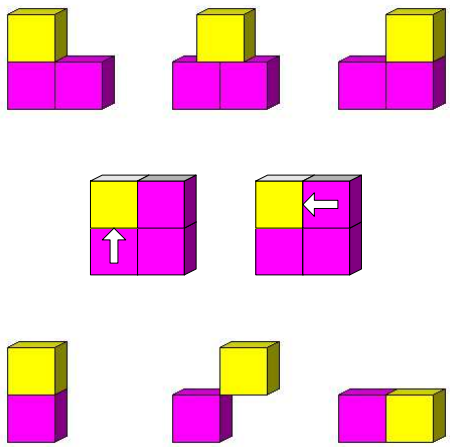


Fig. 1. The basic motions of the abstract module are translation (top), concave transition (center), and convex transition (bottom). The concave transition is not indicated by module motion, rather it is a connection swap indicated by the arrows (the initial connection is to the horizontal surface—after the concave transition the connection is to the vertical surface). The center step in the convex transition may appear to be a difficult pose to emulate in hardware since the cubes only have edge contact. However, hardware systems have been built which support this pose, although only in two dimensions [8], [15].

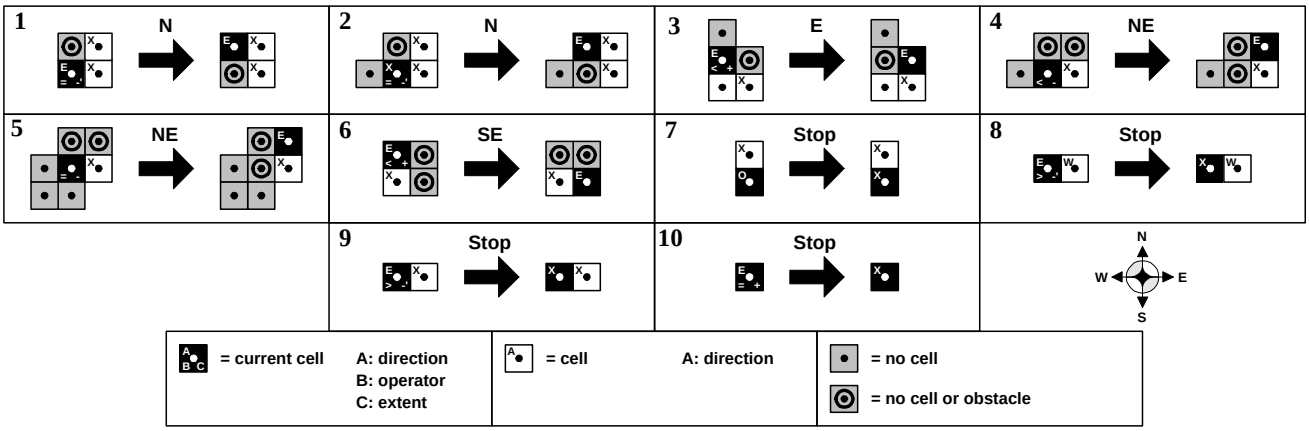


Fig. 2. Partial rule set for building a cube from a flat sheet of modules, where the sheet and the cube have the same y dimension. Only the east rules are shown—all rules other than Rule 7 are duplicated in the west direction, with corresponding changes to the directions and operators in the individual cells of the rules, resulting in a total of 19 rules. Cell variable "A" is the direction state variable which can assume the values $\{N, E, S, W, O, X\}$, where N is north, E is east, S is south, W is west, O indicates any of $\{N, E, S, W\}$ and X indicates no direction, i.e. the cell is not moving. Cell variable "B" denotes the comparison operator applied to the x component of the cell location variable and the cube extents and can assume the values $\{<, =, >\}$. Cell variable "C" indicates which cube x extent is being compared and can assume the values $\{-, +, -', +'\}$, where $-$ is the minimum x extent of the cube, $+$ is the maximum x extent of the cube, $-'$ is the minimum x extent minus one, and $+'$ is the maximum x extent plus one. For example, the preconditions for the current cell in rule 8 are that its direction state is east, and that its x location be greater than the minimum x extent of the cube minus 1. This rule set uses the D_∞ evaluation model.

a) Abstract module: The use of an abstract module allows us to decouple the salient features of a self-reconfiguring module from the implementation dependent features that tend to complicate algorithm design and analysis. We generally represent the module shape as a cube, although our proposed abstraction can be replaced by any geometric structure that supports the formation of lattices. The actuation modalities for the module are the basic motions needed for motion: linear translation, concave transition, and convex transition (see Fig. 1). While no existing three-dimensional module can perform all these motions exactly as our abstract module does, most modules can perform a subset of these motions alone and, with the assistance of other modules, can perform all of them. Module interconnection is face-to-face, e.g. modules can connect when their faces are adjacent and aligned. However, connections are not explicitly simulated—we assume that any face-to-face modules are connected as long as both modules are stationary.

b) Algorithms: Each algorithm employs a set of local rules we implement as a type of cellular automaton. Each rule requires a set of preconditions on the neighborhood of the cell and when activated, causes a change in the system state. The rules are written in the form of productions, with the precondition state on the left and the resulting state, or postcondition, on the right (see Fig. 2). The postcondition is often the movement of a module, but in some cases it is only a change in the internal state in the module. Simple algorithms may not require any internal state, while more complex algorithms may have several local variables for each module. The algorithms presented in this paper were created manually, however we are exploring automated algorithm development.

An important consideration for our algorithms (and our proofs) is the evaluation model used to process the modules. Traditional cellular automata simulators evaluate all cells using their current local states and then update the entire cell array simultaneously. Although it is possible to implement a global clock to synchronize module updates [9], we have chosen to use multiple evaluation models which reflect the expected level of actuation delay in a real system [4].[1] It is worth noting that different evaluation models affect the algorithm rule complexity—more delay generally requires more complex rules. The rule sets presented in this paper use the D_1 and D_∞ evaluation models. The D_1 model allows a delay value of one and the D_∞ model allows arbitrary delay.

c) Correctness: Proof of correctness is an important principle of our approach. Our proofs take the form of logical arguments regarding the possible configurations of module groups [2], [4], and of automated proofs of correctness based on an evaluation of the properties of a constructed graph representing all possible sequences of rule actuations an algorithm for a given initial configuration [10]. In this sense, an initial configuration and a rule set can be viewed as a finite automaton whose graph indicates various properties about the algorithm.

Furthermore we show that simulation results can be combined with machine learning theory to produce a statistical bound on the likelihood that an algorithm is correct. This is done by performing multiple simulations using a fixed rule set and initial configuration, while evaluating modules randomly in each simulation. A specific evaluation order in which rules are executed is referred to as an "activation sequence". The result of many random simulations is an

[1]The amount of actuation delay indicates the level of non-synchronous behavior with regard to rule evaluation time. However, module movement time is instantaneous which eliminates asynchrony due to actuation time.

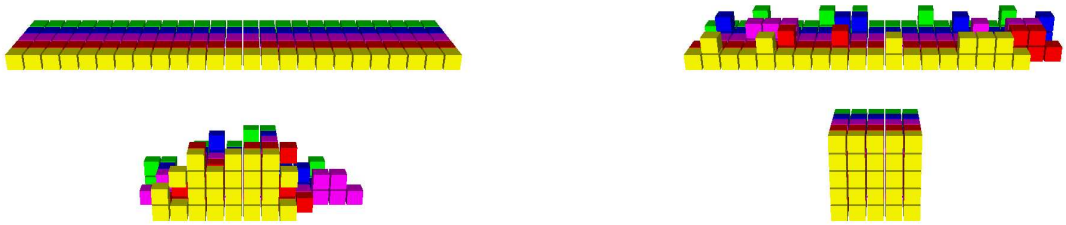


Fig. 3. Four snapshots from a simulation of the cube building rule set in Fig. 2. The initial structure is a 25 x 1 x 5 sheet of modules (the z axis is perpendicular to the page). As the simulation proceeds, modules at the east and west ends of the sheet move toward the center to form a cube. In this example the cube is formed at the center of the x dimension of the sheet, but the rule set functions correctly for any x value of the goal position of the cube along the sheet (the z extents of the cube and the sheet must be aligned).

exploration of the set of all possible activation sequences, which can be used to bound the expected size of the set of erroneous activation sequences.

Our PAC (Probably Approximately Correct) approach can be used to bound the size of the error region, ϵ, with a confidence of δ for a given number of correct, random simulations as follows. Assume that the size of the error region for a given rule set is ϵ. Then, the probability of running n random correct simulations is $(1-\epsilon)^n$. We want to bound this probability by δ, resulting in the following equation

$$(1-\epsilon)^n < \delta. \tag{1}$$

Solving for n yields

$$n > \frac{1}{ln(\frac{1}{1-\epsilon})} ln(\frac{1}{\delta}) \tag{2}$$

and, since $\frac{1}{\epsilon} > \frac{1}{ln(\frac{1}{1-\epsilon})}$ in the range (0,1) the result can be simplified to

$$n > \frac{1}{\epsilon} ln(\frac{1}{\delta}). \tag{3}$$

This provides an estimate on the number of random correct simulations necessary to bound the error region to size of ϵ with a confidence of δ. Thus, if a value of 0.0001 is chosen for both δ and ϵ, the number of simulations required to be 99.99% confident that the size of the error region is no more than 0.01% of the total number of possible activation sequences is, $n > \frac{1}{0.0001} ln(\frac{1}{0.0001}) \approx 92104$.

d) Instantiation: Instantiation refers to the application of generic algorithms to specific hardware systems. This can be done by using meta-modules—groups of real modules which together can perform the basic motion primitives of our abstract module—or by creating "native" module motion sequences which implement the basic motions. The value of instantiation is that the proof of correctness is inherited from the abstract system, and such proofs may not exist for the system using native module motions. Refer to [4] for instantiation examples.

IV. Algorithms

A. Locomotion

Algorithms for locomotion with and without obstacles are described in [2]. The cellular automata approach is well suited to locomotion algorithms, since these algorithms are not fundamentally concerned with the global shape of the robot. Thus, the shape is free to be dynamically altered at the local module level, using local rules. In fact, local conformity of the shape to unknown, rough terrain is an advantage for locomotion. Locomotion algorithm extensions, such as climbing tall obstacles and moving through tunnels, are presented in [4].

B. Assembly

Although our generic distributed approach is well-suited to locomotion tasks, we are interested in exploring whether the cellular automata approach can be used for non-locomotion algorithms such as building specific shapes. Here, the algorithm must be able to control shape formation using only local rules. As our results demonstrate, it is possible to achieve global shape control using only local rules for some shapes.

A key component of our assembly algorithm is that modules know the goal shape, the location of goal shape, and their location.[2] Module location is not difficult to maintain, assuming modules know their location in the initial configuration, since modules move in integral lattice coordinates and can easily update their location as they move. One can imagine a specific module broadcasting the command to build a cube around itself, including the x, y, and z extents of the cube in the message based on its current location. As the message is received by other modules, they can compare their location with the cube extents and determine the direction in which they need to move. Their only motivation would be to place themselves inside the extents of the cube.

A rule set for this algorithm can be seen in Fig. 2. The initial configuration for this algorithm is a planar sheet of n^2 x n modules whose z extents are equal to the z extents

[2]Modules in each layer (x-y plane) need only know the goal shape for that layer, since the assembly algorithm does not move modules out of their initial layer.

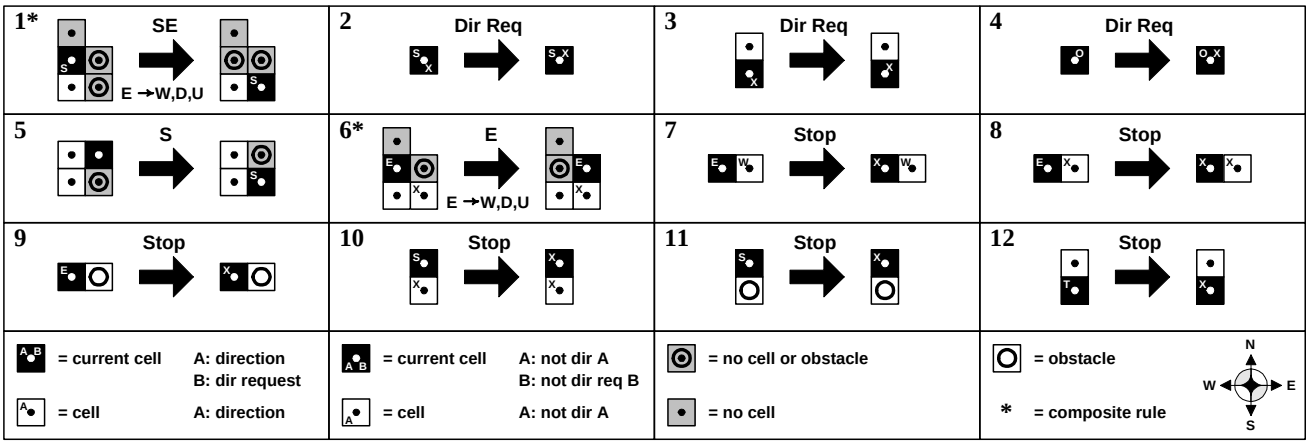


Fig. 4. Partial rule set for dynamic hole repair. Rules are shown for the east direction only—Rules 1, 6, 7, 8, and 9 are duplicated for the west, up, and down directions. Rules 1 and 6 are composite rules—each consisting of 8 individual rules, implementing the constraint that any cell adjacent to the current cell must not be moving south. Fig. 5 illustrates the individual rules for Rule 6. Up and down correspond to $+z$ (out of the page) and $-z$ (into the page) directions, respectively. Cell variable A is the direction state variable which can assume the values $\{N, E, S, W, U, D, O, X, T\}$, where O indicates one of $\{N, E, S, W, U, D\}$, i.e. the cell is moving; X indicates none of $\{N, E, S, W, U, D\}$, i.e. the cell is not moving; and T indicates one of $\{N, E, W, U, D\}$, i.e. the cell is moving but not south. If cell variable A is in the lower left corner of the cell representation it means $\neg A$, i.e. any direction other than A will match the direction variable. Cell variable B is the direction request variable which is used by other cells to request a direction change in a neighbor cell. It can assume the same values as cell variable A, and if it is in the lower right corner it means $\neg B$. The label under the production arrow for rules 1 and 6 indicates a direction request by the moving cell, specifically $R \rightarrow R_1, R_2, R_3$ denotes that any neighbor cells in the R_1, R_2, R_3 directions have their direction request state variable set to R. This rule set uses the D_1 activation model.

of the cube (see Fig. 3). The goal configuration is cube of modules, with n modules in each dimension. Because the sheet and the cube are aligned in the z dimension, no z displacement of modules is necessary—all module movement will be in the x and y dimensions. This restriction allows the use of local module location information only. If modules were not restricted to their initial layers local information would be insufficient determine proper layer placement.

An examination of the rule set reveals that some internal state is used in the modules, specifically the current location of the module, the extents of the cube, and a direction variable which indicates in which direction the module is attempting to move. For example, the north rules (Rule 1 and Rule 2) can only be executed when the module's x location is one less than the minimum x extent of the cube. This means that an eastward-moving module can only move north along the west face of the cube. This restriction prevents modules from moving north inside the cube, which could result in too much vertical development of the shape. As it is, eastward-moving modules can only move east or southeast inside the cube extents which naturally fills horizontal layers one at a time. Note also that the southeast move (Rule 6) is restricted to x locations strictly less than the maximum x extent of the cube. This prevents an eastward-moving module from moving onto the east face of the cube, which would be outside the cube extents. Although Rule 10 implies that any eastward-moving module will stop on reaching the maximum x extent of the cube, due to random rule evaluation it is possible that Rule 6 might be evaluated before Rule 10, requiring that Rule 6 have its own movement restriction. Refer to Section V for experimental results.

Although the above assembly algorithm is correct, it is a bit cumbersome to apply it to different shapes. We have simplified the algorithm to use a simple shape function which indicates whether a given location is inside or outside the goal shape. The rule set has also been slightly modified to allow it to build any filled non-cantilevered convex configuration of modules within each layer (x-y plane).[3] Because each layer is essentially independent, a different three-dimensional convex shape can be built in each layer as long as the layers remain connected, resulting in a large class of feasible goal shapes.

C. Repair

The ability of cellular automata algorithms to emulate flowing fluids suggests that a dynamic structure could be created that would redistribute module locations so as to maintain a continuous membrane of modules. The structure would be multilayered, with the layers parallel to the surface of the structure. Such a structure would be self-sealing—any holes that developed in the structure would be sealed by the reconfiguration of the modules, with the multiple layers of the structure providing the necessary module redundancy to allow holes to be filled (with a resulting reduction in the number of layers over time). A self-sealing structure would be useful in a hazardous environment, for example as the walls of a space station which could dynamically seal any holes due to collisions with foreign bodies, preventing the venting of air. Another possibility is the development of adaptive armor for military vehicles. Currently, a projectile can damage armor in a specific location such that a second hit in that area

[3] A "filled" layer means the layer has no holes, i.e. there are no empty module locations within the module perimeter.

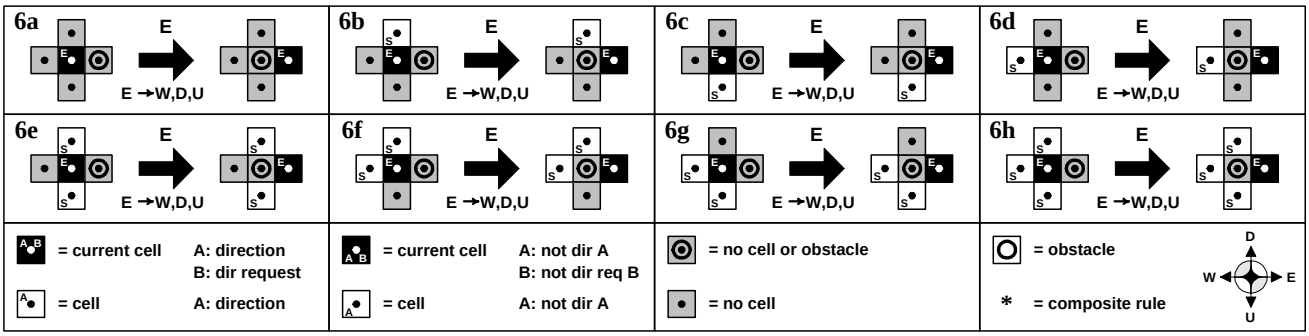


Fig. 5. Individual rules for Fig. 4, Rule 6. These rules enforce the constraint that no cell adjacent to the current cell can be moving south. Because the current rule language does not permit the OR-ing of rule preconditions, eight separate rules are required to test all combinations of the three adjacent cells of the current cell. A similar set of eight rules is used for Rule 1.

will penetrate the vehicle. Adaptive armor composed of self-reconfiguring modules would dynamically redistribute the module locations in order to repair the damage from a previous impact, thus increasing survivability for the occupants.

Figures 4 and 5 show the rule set for hole repair. The idea behind the rule set is that modules without neighbors will move into empty space in order to fill it (see Fig. 6). Specifically, any module in layer i without neighbors in some direction d in both layer i and layer $i-1$, and with no module above it, will perform a convex transition in the Sd direction (Rule 1) and continue to move south (Rule 5) until an obstacle or another module is encountered (Rule 10 and Rule 11). For simulation purposes the layers are parallel to the "floor," which is an implicit obstacle. Unlike the assembly algorithm described above, there is no specific goal shape for this algorithm other than to fill the holes as completely as possible. In this sense the repair algorithm is similar to the locomotion algorithms described in [2], [4].

Rules 1 and 6 are composite rules, composed of eight separate rules which guarantee that no neighbor of the current cell is moving south. This is required to prevent module disconnection. Fig. 5 shows the eight individual rules which comprise Rule 6. Rules 1, 6, 7, 8, and 9 are also duplicated in the west, up, and down directions, resulting in a total of 83 rules. A feature of Rules 1 and 6 is that the moving module submits a direction request to its neighbors. If the proper conditions are met (Rules 2 and 3 are not satisfied), then Rule 4 transfers the direction from the direction request variable to the direction variable. As these modules then begin to move the direction requests cascade to their neighbors in turn, resulting in a mass module movement toward the hole. This is necessary because the size of the hole cannot be determined using local knowledge alone, and therefore as many modules as possible must be moved toward the hole in order to be sure of filling it. A consequence of this is extraneous module movement after the hole is filled, as moving modules continue their motion until they can no longer move in that direction. This is illustrated in Fig. 6 (d) where all the modules in the top layer have migrated toward the hole location. Adding extra rules may permit the motion to be stopped sooner, but again, it may be difficult to precisely determine whether the hole is filled using only local knowledge. If the modules surrounding the hole could communicate to determine the size of the hole prior to moving, it might be possible to limit the number of modules set in motion and to stop module motion after the hole is sealed. Of course, such communication would be outside the scope of the cellular automata paradigm, but it could be implemented as a preprocessing step which sets some internal state in the modules which is then used by the cellular automata algorithm. See Section V for experimental results of hole repair simulations.

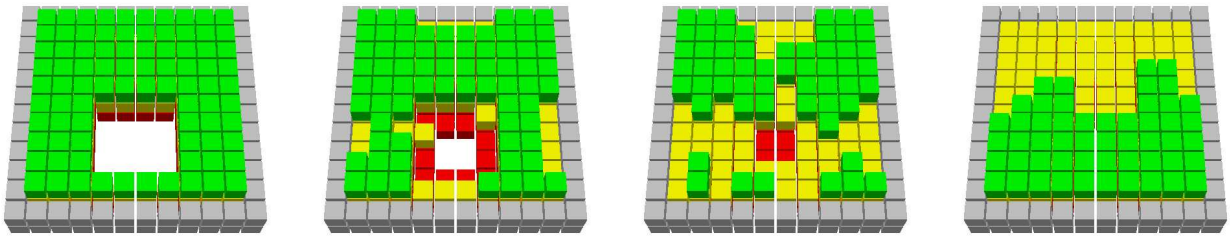
V. Experiments

This section describes the experiments used to demonstrate correctness for the assembly and repair rule sets. The graph analysis experiments use a special mode in our simulator that constructs graphs which represent the finite automaton defined by the rule set and the initial configuration of modules. PAC analysis experiments are simulations of rule set activation on a given initial module configuration. Most experiments were performed on a 2.4GHz Intel Pentium 4 computer running Linux, however in some cases experiments were performed on other machines. The elapsed times for these experiments have been adjusted to estimate the value for a 2.4GHz machine.

A. Assembly

1) Graph analysis: The assembly rule set shown in Fig. 2 is designed to build a cube shape from a flat sheet of modules. Actually, it can build any rectilinear shape, since a single layer can build any rectangle and multiple layers can be stacked to achieve arbitrary width in the z dimension. This rule set uses the D_∞ activation model, and therefore the graph analysis method can be employed to prove the correctness of the rule set for specific instances.[4] Table I shows the results of several graph analysis experiments. These results are for a single layer of modules. Since the rule set does not reference other layers the results can be

[4]The current implementation of the simulator only permits graph analysis of D_∞ rule sets.



(a) (b) (c) (d)

Fig. 6. Four snapshots from a simulation of the hole repair rule set in Figures 4 and 5. The modules are arranged in three 10 x 10 layers, with a 4 x 4 module hole. The perimeter of the cell array is composed of "obstacle" cubes used to contain the group. As the simulation proceeds, the hole is gradually filled by modules surrounding the hole. Note that the experiments described in Section V-B.1 use 12 x 12 module layers.

applied to systems with more than one layer without loss of correctness as long as there is no layer shear.[5] The number of nodes in each graph is slightly less than 2^n, where n is the number of modules. Because the number of nodes is exponential in n, the tractable node count is limited. However, even though larger systems cannot be analyzed using this method, the results for smaller systems provide some confidence that that the algorithm will extend to systems with more modules.

TABLE I

EXPERIMENTAL RESULTS FOR DEMONSTRATING THE CORRECTNESS OF THE ASSEMBLY RULE SET USING THE GRAPH ANALYSIS METHOD.

Size	Nodes	Edges	Activation Sequences	Elapsed Time
2 x 2	13	16	6	$<$ 1 second
3 x 2	66	104	160	$<$ 1 second
2 x 3	69	117	1220	$<$ 1 second
3 x 3	609	1372	409925334	1 second
4 x 3	3756	10159	?	37 seconds
3 x 4	3460	9215	?	33 seconds
4 x 4	31920	103938	?	1031 seconds
5 x 4	279464	1081364	?	110520 seconds

Since the assembly rule set has a definite end state, the graph must not have any cycles since a cycle would imply that the algorithm may never terminate. Also, there must be a single leaf, the desired end state of the building process. It is easy to verify these properties, as the graph can be examined for the presence of back edges which imply cycles and for the existence of a single leaf which has the desired shape.

2) PAC analysis: The PAC analysis method, described in Section III, was applied to the assembly rule set. Here, the running time is polynomial in the number of modules which permits larger module counts. The results are shown in Table II. All iterations were successful and the activation sequences were unique. "Avg. Activation Sequence Length" is the average length of the activation sequences over all iterations. These results are for a single layer of modules. For 100000 unique, correct runs of the assembly rule set we can determine that 99.99% of the total number actuation sequences should be correct with a confidence of 99.99% (7000 runs provides a 99.9% confidence that 99.9% of the actuation sequences are correct).

[5]Layer shear is caused by multiple layers moving at different speeds such that the layers become disconnected [10]. Layer shear is not possible in this assembly task.

TABLE II

EXPERIMENTAL RESULTS FOR DEMONSTRATING THE CORRECTNESS OF THE ASSEMBLY RULE SET USING THE PAC ANALYSIS METHOD.

Size	Iterations	Avg. Activation Sequence Length	Elapsed Time
3 x 3	100000	25.3	0.93 hours
4 x 4	100000	76.1	1.06 hours
5 x 5	100000	181.8	1.66 hours
6 x 6	100000	372.5	6.93 hours
7 x 7	100000	682.4	12.03 hours
8 x 8	100000	1153.3	38.35 hours
9 x 9	7000	1831.3	8.19 hours
10 x 10	7000	2773.0	19.22 hours

B. Repair

1) PAC analysis: The hole repair rule set, as shown in Figures 4 and 5, is designed to fill holes in the structure with modules from the upper layers of the structure (here the layers are in the y dimension instead of the z dimension as in the other rule sets). Since this rule set uses the D_1 activation model, graph analysis cannot be performed, as graph analysis currently only works for rule sets that use the D_∞ activation model. Therefore, only PAC analysis experiments were performed. Table III gives the results for various locations of a 4 x 4 hole in structures with three to five layers. The initial module array is 12 x y x 12, where y is the number of layers. The hole size is 4 x y x 4. All holes have vertical sides. Each experiment consisted of 7000 iterations, and all activation sequences were unique. "Hole Offset" is the offset of the hole from the center of the structure. The values listed under the heading "Successes" are given for the minimum layer not completely filled. For example, a listing of "2: 31" indicates 31 simulations completed without error (disconnection) and the 2nd layer was not completely filled. For these experiments, the number of iterations was reduced to due to the increased simulation time needed for the large number of modules. By the PAC equation (3), 7000 unique, correct

iterations provides a 99.9% confidence that 99.9% of the actuation sequences are correct.

The termination criteria for the hole repair rule set is that there is at least one completely filled layer. Ideally, the hole would be filled by only modules from the top layer, guaranteeing that the resultant structure has as many fully occupied layers as possible. However, as seen in Table III, in some cases the hole is not completely filled. For all experiments the hole is guaranteed to be sealed, but perhaps only with a single layer of modules. Generally, the hole filling is fairly complete with mostly single module vacancies on the next-to-top layer, although in a few cases there was a vacant location in a lower layer.

TABLE III

EXPERIMENTAL RESULTS FOR DEMONSTRATING THE CORRECTNESS OF THE HOLE REPAIR RULE SET USING THE PAC ANALYSIS METHOD.

Layers	Hole Offset	Successes	Avg. Actuation Sequence Length	Elapsed Time
3	(0,0,0)	3: 7000	566.0	7.52 hours
3	(1,0,1)	3: 7000	895.0	12.07 hours
3	(2,0,2)	3: 6969 2: 31	956.1	12.71 hours
3	(3,0,3)	3: 6993 2: 7	905.4	12.07 hours
3	(4,0,4)	3: 7000	860.7	11.51 hours
4	(0,0,0)	4: 6993 3: 7	1529.1	34.24 hours
4	(1,0,1)	4: 6989 3: 11	1375.5	31.57 hours
4	(2,0,2)	4: 5672 3: 1328	1592.1	37.05 hours
4	(3,0,3)	4: 7000	1058.1	24.16 hours
4	(4,0,4)	4: 4975 3: 2025	1859.3	43.45 hours
5	(0,0,0)	5: 5745 4: 1255	1982.5	68.61 hours
5	(1,0,1)	5: 6511 4: 489	1937.2	68.57 hours
5	(2,0,2)	5: 6944 4: 56	1803.4	64.47 hours
5	(3,0,3)	5: 6699 4: 301	1902.6	67.48 hours
5	(4,0,4)	5: 3700 4: 3297 3: 3	2651.8	65.29 hours

VI. CONCLUSION

This paper describes generic distributed algorithms for assembly and repair tasks for modular self-reconfigurable robots. Our approach is based on a cube-shaped abstract module which simplifies algorithm design and analysis. Algorithms for assembling a cube from a flat sheet of modules and for hole repair in a multi-layer module structure are presented. Algorithms are written as sets of rules which only require local information, allowing the algorithms to easily be distributed onto systems with large numbers of modules. Our previous work describes how these generic algorithms can be instantiated onto various hardware systems. Finally, we describe our simulation experiments which demonstrate that the algorithms function correctly.

ACKNOWLEDGMENT

Support for this work was provided through NSF awards IRI-9714332, EIA-9901589, IIS-9818299, IIS-9912193 and EIA-0202789, ONR award N00014-01-1-0675, Intel, and MIT's project Oxygen. We are grateful for this support.

REFERENCES

[1] G. Beni, The Concept of Cellular Robotic Systems, *Proc. of Symposium on Intelligent Control*, 1988.

[2] Z. Butler, K. Kotay, D. Rus, and K. Tomita, Generic Decentralized Control for a Class of Self-Reconfigurable Robots, *Proc. of the IEEE Intl. Conf. on Robotics and Automation*, 2002.

[3] Z. Butler, S. Murata, and D. Rus, Distributed Replication Algorithms for Self-Reconfiguring Modular Robots, *Distributed Autonomous Robotic Systems 5*, Springer-Verlag, 2002.

[4] Z. Butler, K. Kotay, D. Rus, and K. Tomita, Generic Decentralized Locomotion Control for Lattice-Based Self-Reconfigurable Robots, *Intl. Journal of Robotics Research*, to appear.

[5] T. Fukuda and S. Nakagawa, A Dynamically Reconfigurable Robotic System (Concept of a System and Optimal Configurations), *Proc. of International Conference on Industrial Electronics, Control, and Instrumentation*, 1987.

[6] S. Hackwood and J. Wang, Application and Engineering of Cellular Robotic Systems, *Proc. of Symposium on Intelligent Control*, 1988.

[7] J. Heudin, *La Vie Artificielle*, Herms, Paris, 1994.

[8] K. Hosokawa, T. Tsujimori, T. Fujii, H. Kaetsu, H. Asama, Y. Kuroda, and I. Endo, Self-Organizing Collective Robots with Morphogenesis in a Vertical Plane, *Proc. of the IEEE Intl. Conf. on Robotics and Automation*, 1998.

[9] S. Kokaji, S. Murata, H. Kurokawa, and K. Tomita, Clock Synchronization Mechanisms for a Distributed Autonomous System, *Journal of Robotics and Mechatronics*, vol. 8, no. 5, 1996.

[10] K. Kotay, Self-Reconfiguring Robots: Designs, Algorithms, and Applications, Ph.D. Thesis, Dartmouth College, Computer Science Department, 2003.

[11] K. Kotay and D. Rus, Locomotion versatility through self-reconfiguration, *Robotics and Autonomous Systems*, vol. 26, 1999.

[12] J. Kubica, A. Casal, and T. Hogg, Complex Behaviors from Local Rules in Modular Self-reconfigurable Robots, *Proc. of IEEE International Conference on Robotics and Automation*, 2001.

[13] S. Murata, H. Kurokawa and S. Kokaji, Self-Assembling Machine, *Proc. of IEEE International Conference on Robotics and Automation*, 1994.

[14] S. Murata, E. Yoshida, A. Kamimura, H. Kurokawa, K. Tomita and S. Kokaji, M-TRAN: Self-Reconfigurable Modular Robotic System, *IEEE/ASME Trans. on Mechatronics*, vol. 7, no. 4, 2002.

[15] A. Pamecha, C.-J. Chiang, D. Stein, and G. Chirikjian, Design and implementation of metamorphic robots, *Proc. of the 1996 ASME Design Engineering Technical Conf. and Computers in Engineering Conf.*, 1996.

[16] D. Rus and M. Vona, Crystalline Robots: Self-reconfiguration with Unit-compressible Modules, *Autonomous Robots*, vol. 10, no. 1, 2001.

[17] K. Støy, W.-M. Shen and P. Will, Global Locomotion from Local Interaction in Self-Reconfigurable Robots, *Proc. of Intelligent Autonomous Systems 7*, 2002.

[18] J. Suh, S. Homans, and M. Yim, *Telecubes*: Mechanical Design of a Module for Self-Reconfiguring Robotics, *Proc. of IEEE International Conference on Robotics and Automation*, 2002.

[19] K. Tomita, S. Murata, E. Yoshida, H. Kurokawa and S. Kokaji, Reconfiguration Method for a Distributed Mechanical System, *Distributed Autonomous Robotic System 2*, Springer-Verlag, 1996.

[20] C. Ünsal, H. Kiliççöte, and P. Khosla, A 3-D Self-Reconfigurable Bipartite Robotic System: Implementation and Motion Planning, *Autonomous Robots*, vol. 10, no. 1, 2001.

[21] M. Yim, A Reconfigurable Modular Robot with Many Modes of Locomotion, *Proc. of JSME Conference on Advanced Mechatronics*, 1993.

[22] E. Yoshida, S. Murata, H. Kurokawa, K. Tomita and S. Kokaji, A Distibuted Method for Reconfiguration of 3-D Homogeneous Structure, *Advanced Robotics*, vol. 13, no. 4, 1999.